Cairo University ECO Racing Team

Motor Control Team

Pre-Interview Task Submission

Name: Amr Ahmed Sayed

Department and Year: Electrical Power and Machines Engineering

Mobile phone number: 01154341558

E-mail: amryoyo445@gmail.com

Project Drive Link: **CLICK HERE**

Question 1

DC Motor vs. BLDC Motor

Feature	Brushed DC Motor	BLDC Motor (Brushless DC)
Construction	Rotor with windings, stator with magnets; mechanical brushes and commutator	Rotor with permanent magnets, stator with 3-phase windings; no brushes or commutator
Operation	Brushes switch current in rotor, creating rotating magnetic field	Electronic controller energizes stator phases based on rotor position
Commutation	Mechanical (brush/commutator)	Electronic (via controller + sensors or estimators)

BLDC vs. PMSM

Feature	BLDC	PMSM
Back EMF	Trapezoidal	Sinusoidal
Stator Windings	Often concentrated	Typically distributed
Typical Control	Six-step (trapezoidal) commutation	FOC or sinusoidal control
Sensor Requirement	Hall sensors (basic)	Encoder/resolver (more precise)
Key Difference	Commutation waveform & smoothness	PMSM allows smoother torque via FOC

Six-Step Commutation (BLDC)

- How it works:

Divides the 360° electrical cycle into 6 sectors. Only two motor phases are energized at a time based on rotor position.

- Hall Sensors Role:

Detect rotor position every 60° electrical. Tell controller which two phases to energize next.

- Advantages:

Simple, low-cost, easy implementation, works well with Hall sensors.

- Limitations:

Torque ripple, less efficient at low speed, more noise, rougher motion compared to FOC

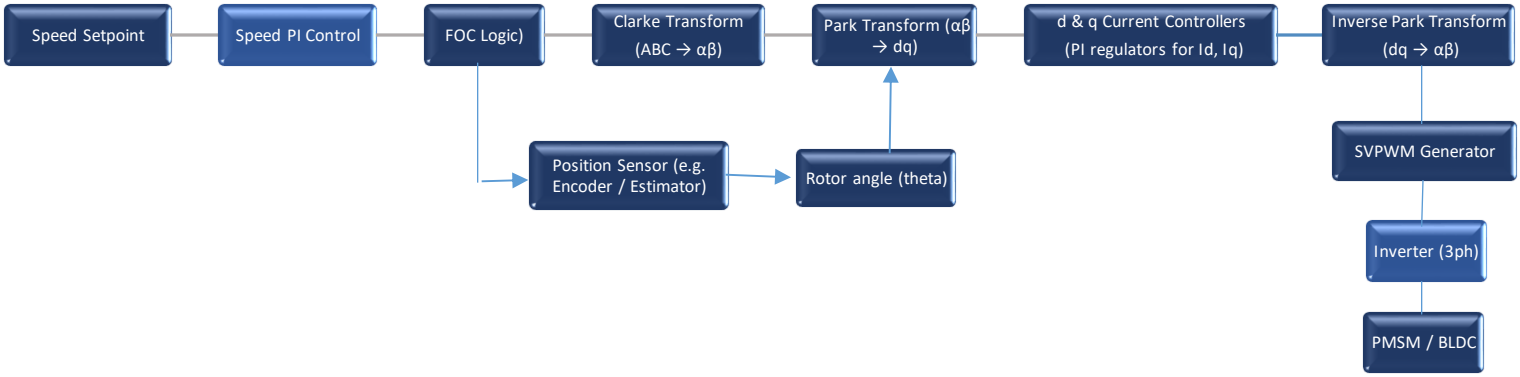
PWM vs. SPWM vs. SVPWM

Type	Description	Impact
PWM	Basic pulse switching, square wave	High harmonics → more losses & ripple
SPWM	Sinusoidal modulation of duty cycle	Smoother output, less ripple, better efficiency
SVPWM	Uses vector theory to synthesize waveform	Up to 15% better DC bus utilization, lower torque ripple, best performance

SVPWM Advantage:
Improves voltage utilization and smoothness, especially in high-performance drives.

Field Oriented Control (FOC)

Transforms 3-phase currents into a rotating d-q frame. Separates flux (Id) and torque (Iq) control like in DC motors.



Explanation :

- We start with a **target speed**, and compare it to actual speed → pass through a **PI controller** to get how much torque we want (I_{q_ref}).
- The motor phase currents (I_a, I_b, I_c) are converted using the **Clarke & Park transforms** into I_d (flux) and I_q (torque).
- We use two **PI controllers** to control Id and I_q (usually we set I_d = 0 to ignore flux).
- Then we use **Inverse Park** to convert control signals back to the 2-phase αβ frame.
- The output goes into **SVPWM**, which gives smoother and more efficient switching than normal PWM.
- The **inverter** drives the motor phases, and the feedback loop continues.

Pros:

- Smooth torque
- High efficiency
- Wide speed control
- Dynamic response

Cons:

- Needs precise rotor position
- More complex computation
- Needs controller tuning

Advantage over Trapezoidal:

Dramatically reduces torque ripple and improves performance at low speeds.

MATLAB/Simulink for Motor Control

- Role in Model-Based Design:

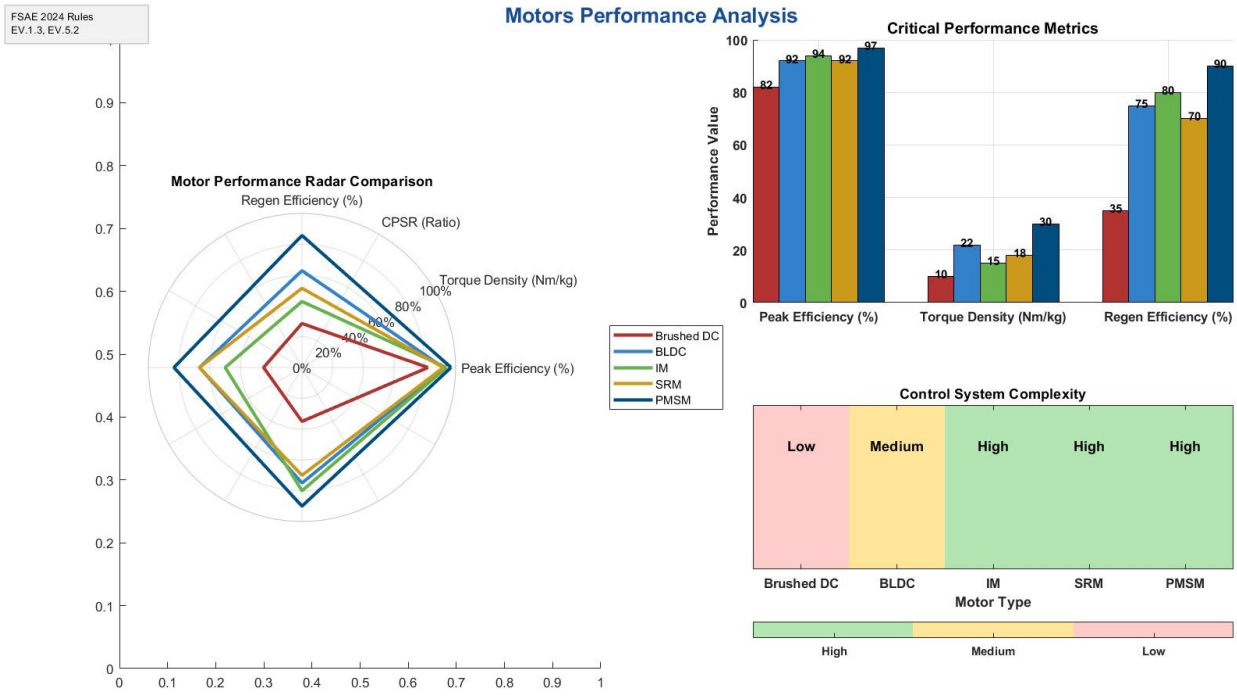
Simulink allows full system simulation (motor, sensors, inverter, …etc) before real hardware is used.

- Simulating BLDC with Hall Sensors:

1. Build motor and inverter model using Simscape Electrical.
2. Simulate Hall sensors using rotor position output.
3. Add speed **PI controller**.
4. Implement 6-step commutation logic based on Hall inputs.
5. Generate PWM signals.
6. Close the loop: **speed feedback → PI → commutation → motor.**
7. Analyze system response, torque ripple, and efficiency.

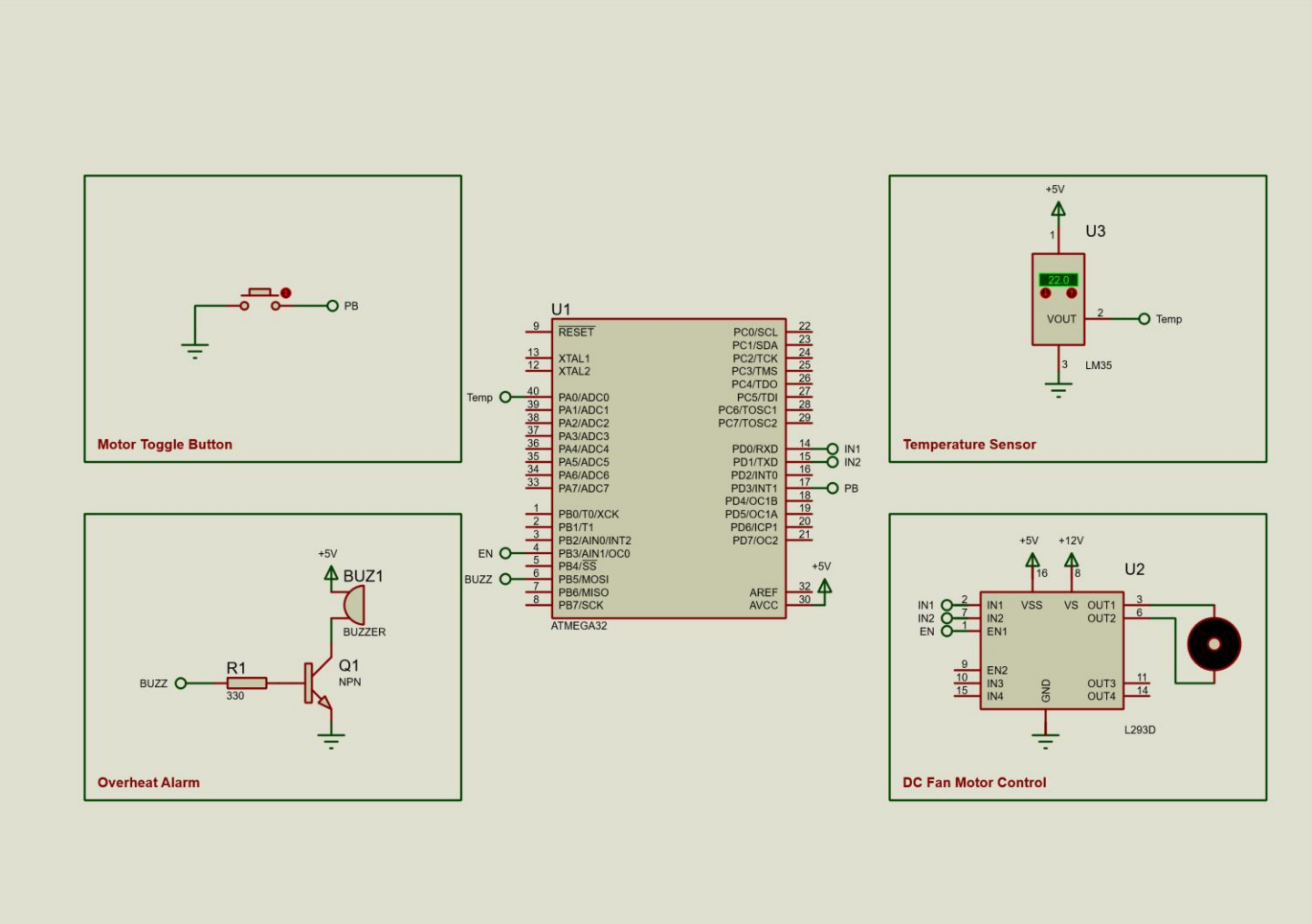
- Extra:

You can also auto-generate code **(Embedded Coder)** for deployment on a real-time controller or microcontroller



This chart summarizes five electric motor types—**Brushed DC, BLDC, Induction Motor (IM), Switched Reluctance Motor (SRM),** and **PMSM**—comparing their **torque density, peak efficiency, efficiency across operating regions**, and **control system complexity**. PMSM offers the highest torque density and efficiency, but requires advanced control (e.g., FOC). Brushed DC is simplest to control but least efficient. SRM has moderate efficiency with high control complexity, while BLDC and IM provide a balance between performance and control effort.

Question 2



Project Overview:

This project implements an intelligent fan control system using the ATmega32 microcontroller. The system reads ambient temperature via an LM35 sensor and adjusts the speed of a DC fan using PWM based on the measured temperature. A push button toggles the motor operation via an external interrupt. If the temperature exceeds 75°C, an overheat warning is triggered through a buzzer using a transistor switch. The fan is driven by an L293D H-Bridge IC to allow efficient speed control.

What the Code Does

1. **Reads Temperature:**
    - The LM35 temperature sensor sends an analog voltage to the microcontroller (ATmega32)
    - The microcontroller reads this voltage using its ADC (Analog-to-Digital Converter) to get the temperature value
2. **Controls Fan Speed:**
    - If the temperature is between 25°C and 75°C, the code increases the fan speed gradually using PWM (Pulse Width Modulation)
    - The hotter it gets, the faster the fan spins
3. **Turns On a Buzzer (Overheat Warning):**
    - If the temperature goes above 75°C, the buzzer is turned on to alert about overheating
4. **Uses a Push Button to Toggle the Fan:**
    - You can press a push button to turn the fan ON or OFF using an external interrupt
5. **Keeps Running Automatically:**
    - Once started, the system continuously checks the temperature and adjusts the fan and buzzer automatically

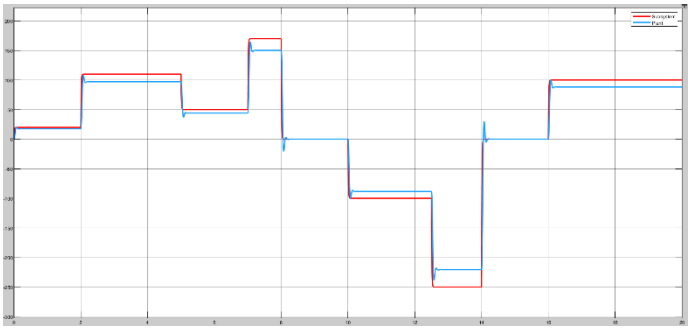
Challenges Faced and How I Solved Them

1. **First-Time Using Proteus:**
    - Problem: This was my first time working with Proteus, and I had to learn the software quickly in order to complete the simulation in time
    - Solution: I watched quick tutorials and explored the interface myself to become familiar with placing components, wiring, and configuring the microcontroller
2. **HEX File Not Loading in ATmega32:**
    - Problem: After building the project, Proteus was unable to load the .hex file correctly, even after uploading it to the ATmega32
    - Solution: I discovered that the issue was caused by a conflict with the VSM Studio programmable file. I solved it by removing the VSM Studio path from the Program File section, so Proteus would only use the correct (.hex) file for simulation.
3. **Incorrect Power Configuration:**
    - Problem: Initially, the circuit did not work because of incorrect or missing power connections, especially on the microcontroller and driver ICs.
    - Solution: I referred to online videos and documentation (e.g., YouTube tutorials) to properly configure power pins and voltage sources for all components.

Question3

Controller Selection and Reasoning

Before designing the controller, I ran the simulation without any controller to see how the motor (plant) behaves on its own. I connected the plant output to a Scope and compared it with the reference signal (stair input).



What I observed:

- The output followed the general shape of the reference, but there was always a gap — meaning the system had a steady-state error.
- The motor was stable (no oscillations), and the response was smooth
- However, the response wasn't fast enough, and it didn't fully reach the desired speed in time.

From this, it was clear that some form of control is needed to improve both accuracy and speed.

Why I chose a PID Controller:

The task asked for:

- Minimal steady-state error (Requires integral action)
- Fast rise and settling times around 0.5 seconds (Needs derivative action)
- Limited or no overshoot (Careful tuning needed)

I first thought of using a PI controller, since it can remove steady-state error. But after looking at the sharp changes in the stair input and the need for quick reaction, I realized that using a PID would give better performance overall. Adding the derivative term helps reduce overshoot and improves how fast and smoothly the system reacts, especially when the reference jumps suddenly.

Final decision:

I went with a PID controller to make sure the system tracks the reference accurately, quickly, and without big overshoots — just like the task required.

Tuning Method: Manual Tuning

I used manual tuning instead of the Ziegler–Nichols method. Since the input is a stair generator (not a sine or constant signal), Ziegler–Nichols was not appropriate.

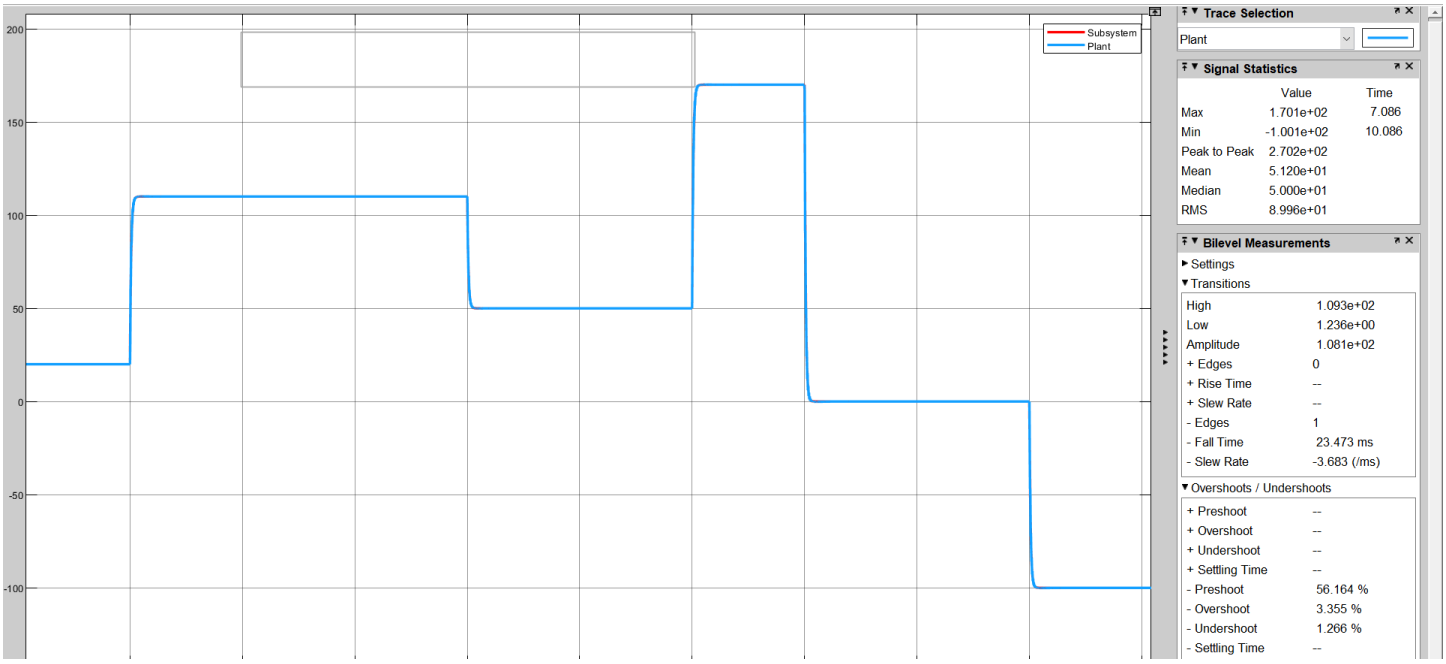
Tuning Process:

- I started with a proportional gain (P) to make the system respond faster.
- Then I added an integral gain (I) to eliminate the steady-state error.
- Finally, I added a small derivative gain (D) to reduce overshoot and improve the response to sharp changes in the reference.

Final PID Values

After several iterations and scope analysis, the best performance was achieved with:

P = 86.4 | I = 1440 | D = 2.63



Performance Evaluation

I compared the reference and output on the scope. The final response showed:

- **Overshoot:** ~3.3%
- **Undershoot:** ~1.3%
- **Settling Time:** ~23 ms (well below 0.5s)
- **Steady-State Error:** ≈ 0%

These results meet all the required specs.

Metric	Value	Target
Rise Time	0.086s	≈0.5s
Settling Time	0.0235s	≈0.5s
Overshoot	3.355%	<10%
Steady-State Error	<0.1%	Minimal
Peak Tracking	170.1/170	Exact

Conclusion

The PID controller successfully made the motor output follow the reference signal with high accuracy and speed, and without severe overshoot. Manual tuning allowed precise adjustments based on real-time simulation feedback.